

Security and Privacy in a Networked World

Even casual users of technology collect, transform, store, and share personal data on networked devices and services. This creates substantial security and privacy challenges for system designers. Social networking sites, such as Facebook, must negotiate conflicting requests, made by different users and advertisers, to share or keep private particular message posts or images. Mobile platforms, such as Google's Android, must—or at least should—provide controls ensuring sensitive information, including location traces and recorded audio, is not disclosed without user knowledge. Finally, cloud storage services, such as Apple's iCloud, and information processing services, such as Mint.com's personal finance website, must implement software that handles private data in a manner consistent with stated terms of service. My research is about translating such high-level *security goals* into precise *policies* and mechanically enforcing policies in systems.

A pair of related problems impede the construction and verification of secure systems. First, English-language security and privacy goals are difficult or impossible to express precisely in current policy configuration tools or languages, and consequently, developers and architects are forced to select policies that don't match their security goals. Second, even when useful security policies can be specified, the sound enforcement of such policies is costly in a variety of ways: manual enforcement disciplines can be difficult to learn and error-prone to maintain, static analyses can be computationally hard, and dynamic checks can be prohibitively slow.

My research seeks to answer the following two questions in a variety of settings.

- *How can users represent their security goals as precise policies, as statements that are neither overly conservative nor overly permissive?*
- *What automated mechanisms can best enforce policies drawn from a given universe of policies?*

I address these questions by building systems in which policies are represented in terms of flexible, expressive objects with clear semantics, and in which the semantic framework provides abstractions necessary to clearly specify the behavior of enforcement mechanisms. This approach requires careful co-design of policy and mechanism, where the feasibility of enforcement informs policy design and where the clarity and utility of policy semantics inform mechanism design. My research has applied these principles to two broad and related challenges: improving the trustworthiness of system implementations, and reasoning directly about different sorts of security-relevant data handled by systems. These two challenges are discussed below.

Secure System Implementation via Information Flow and Access Control

If a developer's security goals can not be represented as an enforceable policy, security mechanisms may allow some undesired system behaviors, forbid some desired behaviors, or both. I address the challenges of expressing security goals as formal policies and of reducing the cost of policy discovery and enforcement.

Cryptographically-enforced information flow types. Program inputs may come from a variety of sources, including sensor devices, the internet, and direct user input. It is often desirable to ensure that that systems maintain *confidentiality* and *integrity* properties. Confidentiality segregates private inputs, such as a Twitter password, from public outputs, such as a shared stream of tweets. Integrity segregates untrusted inputs, such as keyless entry signals received by a car, from trusted outputs, such as device requests that

operate vehicle brakes. *Information flow* techniques enforce these properties by tagging inputs with security labels, propagating these labels through a program so that they describe the contents of variables or values, and checking labels to prevent violations of confidentiality or integrity. Most previous work on information flow focuses on what happens only within a program, using ad-hoc or unsafe techniques to protect data at system edges. I’ve designed principled mechanisms that automatically use cryptography to maintain security at system boundary points and that address different information flow models [VZ07, Vau09].

For example, my work on the Aura security-oriented programming language introduces novel type-based techniques for enforcing confidentiality [Vau11]. Type `String for Alice` labels a value as only readable by principal Alice, and such values are automatically encrypted before, for example, transmission on the network. Encrypted values confound standard type-checking techniques by obscuring the structure of terms. Aura’s type system meets this challenge using unconventional information about statically and dynamically known encryption keys, as well as facts about ciphertexts generated at a host. We validated this design with machine-checked *soundness* and *noninterference* proofs.

Inference of information flow policies. The information-flow analyses described above are, broadly speaking, policy checkers. Given a policy and program the analysis answers *yes, the program certainly obeys policy*, or *no, the program might admit unsafe behaviors*. The use of policy checkers requires effort up-front to write down policies and to refactor or correct programs so that the analysis answers *yes*.

With Stephen Chong of Harvard University, I investigated information-flow policy inference, toward the goal of lowering analysis costs. We designed and built a tool that takes a minimally-annotated Java program as input and outputs a policy met by that program, using an interprocedural object-sensitive dataflow analysis [VC11]. Inferred policies are sound, meaning that all existent information flows are reported, and are intended to be close to a most informative “best” policy. A programmer can refactor or annotate code if the inferred policy does not meet his security goals. This is intended to foster a process of iterative refinement, where weak security guarantees may be had for little effort, and stronger guarantees may be found via more interaction with the inference engine. We validated this tool by successfully analyzing our own Java programs and programs written by others for the JIF [MCN⁺99] information flow system.

Fine-grained access control for mobile devices. *Access control* policies constrain when a program is allowed to access sensitive resources such as a mobile phone’s camera. With colleagues including Jeffrey Foster from the University of Maryland and Todd Millstein from UCLA, I defined a layer of fine-grained access-control permissions on top of the existing Android operating system and adapted Marketplace apps to use these permissions [JMV⁺11]. We introduced new libraries and permissions that, for example, grant access to a single network domain (e.g., `flickr.com`), refining the built-in permission allowing arbitrary internet access. We used binary rewriting to retrofit 19 popular apps, each with more than one million downloads, to use the new permissions with little or no change to user experience.

Reasoning with and about Trusted Data via Audit and Provenance

While access control or information flow policies describe properties of programs, it is equally important to consider security and reliability properties of data itself. Studying *provenance*—the origin and evolution of data—and *audit*—the analysis of past system states based on persistent data—is important for building systems that interact with external data, that can be analyzed in the event of anomalous behavior, and that have small trusted computing bases. Audit is particularly necessary in systems where the cost of undesirable access denials is high. For example, a medical records system should allow otherwise unauthorized clinicians to access patient records to allow treatment in emergency situations, while maintaining logs enabling abuse to be detected later and handled through legal, economic, or social means.

Evidence-based audit for access control. With my collaborators at Penn I integrated access control via *authorization logic* [JVM⁺08] with automatic audit logging [VJMZ08] in the Aura programming language. In Aura, access to security-sensitive resources is only permitted when an appropriate logical proof explicates why access should be granted. Such proofs are logged, and can be examined later to give an unforgeable account of why access was allowed.

Aura proofs comprise both rules of inference and embedded, digitally-signed messages attesting to requests made by principals. Logged proofs may be efficiently checked for validity by verifying digital signatures and confirming that inference rules are composed correctly. Storing verifiable, cryptographic logs reduces the size and complexity of the system's trusted computing base (that is, security rests on fewer, weaker assumptions), and provides a basis for trustworthy audit mechanisms.

Principled data transformation. Changes to a database view may not correspond uniquely to changes on the underlying database. Thus any tool that propagates modifications in a view back to the database requires guidance to select a single, best database update. Guiding updates with ad-hoc code makes it easy to introduce vulnerabilities near the data storage layer and obscures the provenance relations between database and view records. With Aaron Bohannon and Benjamin Pierce, I defined a bidirectional query language, based on relational algebra, in which expressions both define views and guide updates [BPV06]. We proved well-typed update policies are, in a precise sense, well-behaved.

Relatedly, with Chris Skalka from the University of Vermont and Stephen Chong, I developed a novel watermarking scheme for annotating ecological data sets with provenance metadata [CSV10]. The conventional watermarking threat model aims to prevent an adversary from removing metadata without perceptibly changing (e.g.) an image. In contrast, we consider honest scientists who would prefer to retain provenance information, so long as workflows and scientifically meaningful aggregates remain unchanged.

Audit-supported information flow. In ongoing work with with Eric Griffis and Todd Millstein, I describe and implement a prototype Personal Data Vault system [GVM11]. This system provides users with a central point of control for private data, such as GPS-recorded location traces or user-submitted health information, and allows third-party service providers, such as advertisers or physicians, to interact with this data subject to sandbox-enforced constraints. Third-party and built-in plugins are written as side-effect free Lua scripts, and can be composed into interesting data filters. We implemented a sandboxed version of the Lua interpreter and, to bound information leaks a priori, a lightweight static analysis. Additionally we built tools that support counterfactual audit: plugins may be run safely with a mix of real and simulated data, allowing invariants to be checked on historical system states and in hypothetical corner-cases.

Looking Forward: Foundations and Applications for Principled Policies

My research has shown that it's possible to build systems that enforce expressive, user- or developer-selected security and privacy policies. Along the way I hit surprising roadblocks. Current foundations are not general enough to analyze techniques used in practice, enforcement mechanisms can be overly specialized or ad hoc, and conventional approaches to policy languages don't accommodate the security goals relevant to modern systems. In my research, I will strive to address these issues, investigating the basic principles of secure system construction and applying these principles to real systems.

Policies for compositional systems. While the gap between desired and enforceable policy can be narrowed by making permissions fine-grained, this alone is not sufficient to close it. For instance, consider a mobile app that scans barcodes and performs online price comparisons. This app must access both a user's camera and the internet, a dangerous set of permissions. Standard techniques do not distinguish between

barcode data (less sensitive) and photograph images (more sensitive), because both originate from a single source, the camera. Indeed, because the camera produces full photographs as atomic units of data, it's unclear how to slice a "capture image" permission into finer units suitable for making the relevant, data-centric distinctions. As another example, the set of messages viewable to a Facebook user is controlled by privacy settings and actions made by several principles, and it's the distributed nature of this process—not the policy language's granularity—that makes reasoning about privacy challenging. In both cases natural security goals depend on composition in the system, of data and algorithm in the case of barcodes, and of disparate actions of friends in the case of Facebook messages. I plan to design and build systems in which such policies can be described, enforced, and audited.

Language-based security for modern programming models. A successful line of research has used programming languages techniques, such as type systems, to address security problems. However these approaches don't always scale to work with real systems. I plan to investigate the security properties of modern systems, considering features that have received little attention from the language-based security community. In particular, I look forward to investigating popular alternative programming models, including domain specific languages, such as Map-Reduce, and relaxed memory models, such as total store ordering, which pose new security and privacy challenges. For each such programming model, I plan to investigate endemic sources of vulnerabilities, formally describe sound or partial mitigation techniques, and implement a full-scale mechanism to enforce policies. By integrating security into the development process at the language level, I believe we can prevent many classes of vulnerabilities at reasonable cost.

Security via reduction to known analyses. I plan to look for alternate formulations of security properties that are, in part, reducible to well-established patterns of analysis and automated reasoning. I hypothesize that these latter techniques can be leveraged to produce effective security and privacy analyses and enforcement mechanisms. As a first step, I will investigate enforcing information flow policies with probabilistic reasoning techniques. The Geiger-Paz-Pearl axioms of independence were developed to describe when information learned about one random variable does not impart information about another, and strikingly, these axioms are also modeled by *nondeducibility*, a way to describe information flow in systems [MN10]. I plan to exploit this duality to build information-flow analyses based on efficient tools, such as *belief networks*, developed for probabilistic reasoning. In the long run I believe that recasting security challenges, when appropriate, as variants of understood problems will provide a means to mitigate vulnerabilities as well as a means to clarify the meaning of security properties.

Foundational semantics of security mechanisms. Many deployed security techniques are not yet well understood theoretically; this deprives of us a principled way to evaluate, compare, and refine these techniques. As an example, consider simple data *tainting* schemes that tag values in a running program with security metadata, and that dynamically halt the program if it attempts an insecure assignment. While used in the wild and considered a best practice for secure Perl programming [Per11], tainting is poorly understood theoretically. Indeed, when examined using standard techniques, it is found to give no security advantage—a result that contradicts experience. Tainting is just one example of a partial enforcement mechanism, one that mitigates, but does not completely eliminate, a class of security vulnerabilities; other examples include forms of system call interposition and signature-based malware detection. I plan to study foundations for such partial security mechanisms and to evaluate existent and novel techniques in this light. This will require breaking away from the conventional theoretic view that security is a binary proposition and developing more expressive metrics for system security. Similarly, and in contrast to prior work on the structure of audit logs, I plan to investigate the interpretation and the semantics of the audit process. In the long term I plan to use such new foundations to evaluate and improve practical security tools and trusted systems.

References

- [BPV06] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: A language for updatable views. In *Proc. Principles of Database Systems (PODS)*, pages 338–347, 2006.
- [CSV10] Stephen Chong, Christian Skalka, and Jeffrey A. Vaughan. Self-identifying sensor data. In *Proc. Information Processing in Sensor Networks (IPSN)*, pages 82–93, 2010.
- [GVM11] Eric Griffis, Jeffrey A. Vaughan, and Todd Millstein. Transducer-based personal data vaults: A principled architecture for controlled data sharing. In review, 2011.
- [JMV⁺11] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Nikhilesh Reddy, Yixin Zhu, Jeffrey S. Foster, and Todd Millstein. Dr. Android and Mr. Hide: Fine-grained security policies on unmodified Android. In review, 2011.
- [JVM⁺08] Limin Jia, Jeffrey A. Vaughan, Karl Mazurak, Jianzhou Zhao, Luke Zarko, Joseph Schorr, and Steve Zdancewic. Aura: A programming language for authorization and audit. In *Proc. International Conference on Functional Programming (ICFP)*, pages 27–38, 2008.
- [MCN⁺99] Andrew C. Myers, Stephen Chong, Nathaniel Nystrom, Lantian Zheng, and Steve Zdancewic. Jif: Java information flow. Project webpage located at <http://www.cs.cornell.edu/jif/>, 1999.
- [MN10] Sara Miner More and Pavel Naumov. An independence relation for sets of secrets. *Studia Logica*, 94(1):95–107, February 2010.
- [Per11] Perlsec: Perl security. Section of the Perl language reference; located at <http://perldoc.perl.org/index.html>, 2011.
- [Vau09] Jeffrey A. Vaughan. *Aura: Programming with Authorization and Audit*. PhD thesis, University of Pennsylvania, Philadelphia, 2009.
- [Vau11] Jeffrey A. Vaughan. AuraConf: A unified approach to authorization and confidentiality. In *Proc. Workshop on Types in Language Design and Implementation (TLDI)*, pages 45–58, 2011.
- [VC11] Jeffrey A. Vaughan and Stephen Chong. Inference of expressive declassification policies. In *Proc. IEEE Symposium on Security and Privacy (Oakland)*, pages 180–195, May 2011.
- [VJMZ08] Jeffrey A. Vaughan, Limin Jia, Karl Mazurak, and Steve Zdancewic. Evidence-based audit. In *Proc. IEEE Computer Security Foundations Symposium (CSF)*, pages 177–191, 2008.
- [VZ07] Jeffrey A. Vaughan and Steve Zdancewic. A cryptographic decentralized label model. In *Proc. IEEE Symposium on Security and Privacy (Oakland)*, pages 192–206, 2007.